Final Project - Instruction Pipeline Simulation

Michael Del Rose
Technical Paper – Vetronics (In-House)

Report Documentation Page

Form Approved
OMB No. 0704-0188

1. REPORT DATE **02 DEC 2000**	2. REPORT TYPE **N/A**	3. DATES COVERED **-**
4. TITLE AND SUBTITLE **Final Project - Instruction Pipeline Simulation**		5a. CONTRACT NUMBER
		5b. GRANT NUMBER
		5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S) **Michael Del Rose**		5d. PROJECT NUMBER
		5e. TASK NUMBER
		5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **US Army RDECOM-TARDEC 6501 E 11 Mile Rd Warren, MI 48397-5000**		8. PERFORMING ORGANIZATION REPORT NUMBER **14203**
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)		10. SPONSOR/MONITOR'S ACRONYM(S) **TACOM/TARDEC**
		11. SPONSOR/MONITOR'S REPORT NUMBER(S) **14203**
12. DISTRIBUTION/AVAILABILITY STATEMENT **Approved for public release, distribution unlimited**		
13. SUPPLEMENTARY NOTES		
14. ABSTRACT		
15. SUBJECT TERMS		

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT **unclassified**	b. ABSTRACT **unclassified**	c. THIS PAGE **unclassified**	**SAR**	**18**	

Standard Form 298 (Rev. 8-98)
Prescribed by ANSI Std Z39-18

1.0 Introduction

Writing a simulation of an instruction pipeline has given me a good idea of what goes into the logic of it. The simulation I wrote goes through 2 types of pipelines, a 4 stage pipeline and a 5 stage pipeline. I choose to write the logic of the pipeline in Visual Basic, but I use access to keep the simulation graphical.

At the beginning of this project I was not familiar with Visual Basic and only moderately familiar with Access. I now feel that I can set up most types of simple databases with Access and Visual Basic.

2.0 Using the Instruction Pipeline Simulation

The simulation is modeled after figure 2.1.

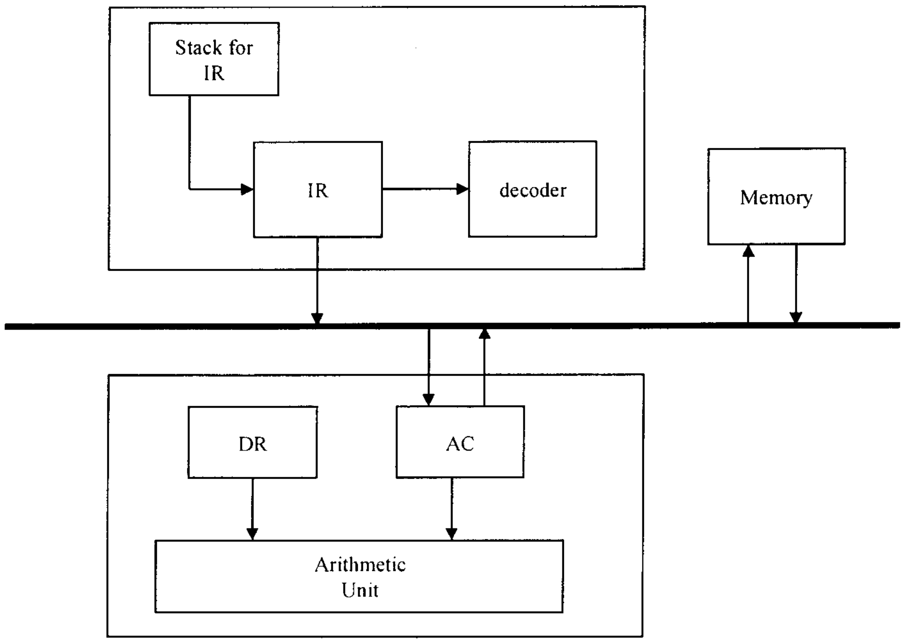


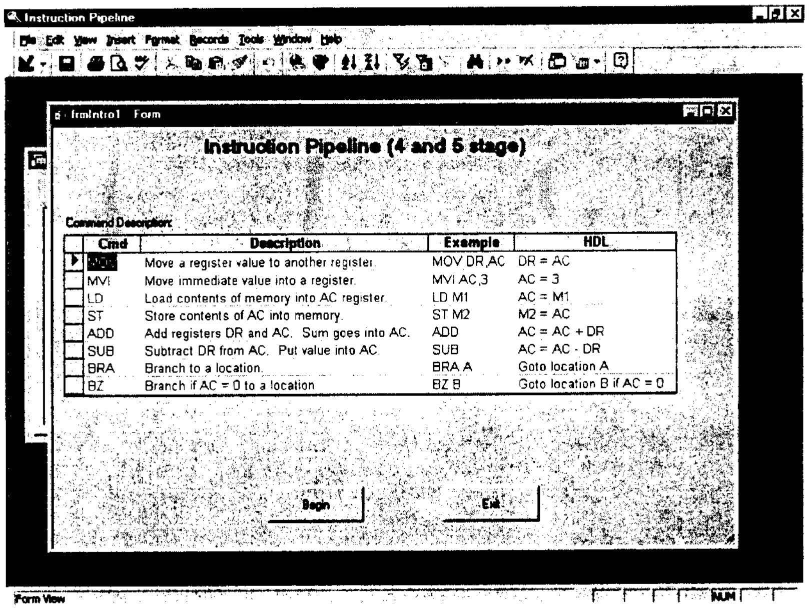
Figure 2.1

It is a load store architecture where only LD and ST commands can access the memory. There are two memory locations (M1 and M2). There are also two registers (AC and DR). The accumulator register (AC) is the only way data can get into or out of the ALU. The other register is called the data register (DR).

The simulation starts off with an introductory screen explaining the available commands (see figure 2.2). These commands are MOV (move register to register), MVI (move immediate value to register), LD (load memory to register), ST (store register contents into memory), ADD (add two registers), SUB (Subtract two registers), BRA (branch unconditionally), and BZ (branch if AC = 0).

The "Begin" button on figure 2.2 will take you to the form for entering the program (see figure 2.3). The "Exit" button will close the simulation.

Notice that in figure 2.3 you have columns where you enter in your program. The columns represent the items that are important in the simulated assembly language. The "Location" column is used to distinguish a part of the program that you might want to jump to. The "Command" column is the command that you want to execute. The "Operand1" and "Operand2" columns are where you put in the operands that go with the command.

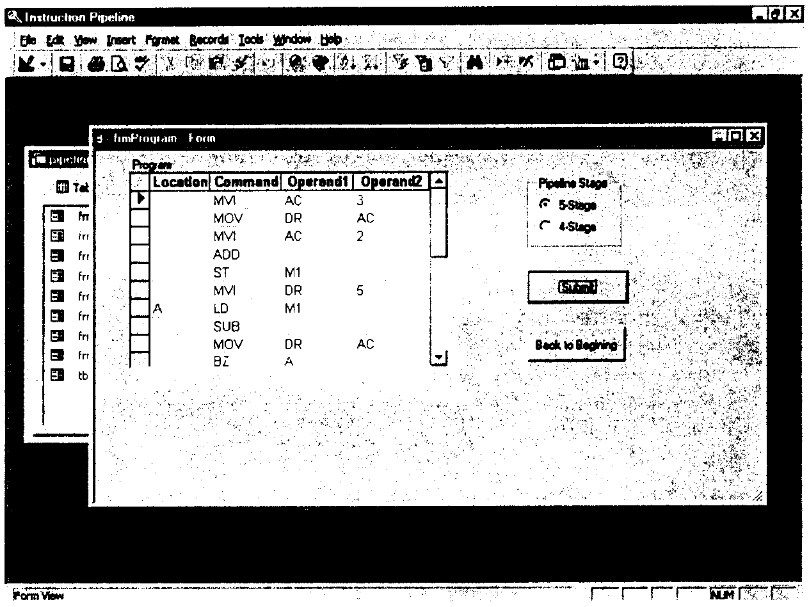


Instruction Pipeline (4 and 5 stage)

Command Description:

Cmd	Description	Example	HDL
MOV	Move a register value to another register.	MOV DR,AC	DR = AC
MVI	Move immediate value into a register.	MVI AC,3	AC = 3
LD	Load contents of memory into AC register.	LD M1	AC = M1
ST	Store contents of AC into memory.	ST M2	M2 = AC
ADD	Add registers DR and AC. Sum goes into AC.	ADD	AC = AC + DR
SUB	Subtract DR from AC. Put value into AC.	SUB	AC = AC - DR
BRA	Branch to a location.	BRA A	Goto location A
BZ	Branch if AC = 0 to a location	BZ B	Goto location B if AC = 0

Figure 2.2



Program

Location	Command	Operand1	Operand2
	MVI	AC	3
	MOV	DR	AC
	MVI	AC	2
	ADD		
	ST	M1	
	MVI	DR	5
A	LD	M1	
	SUB		
	MOV	DR	AC
	BZ	A	

Pipeline Stage
[✓ 5-Stage]
4-Stage

Figure 2.3

For example, suppose you want to enter in the following assembly language program:

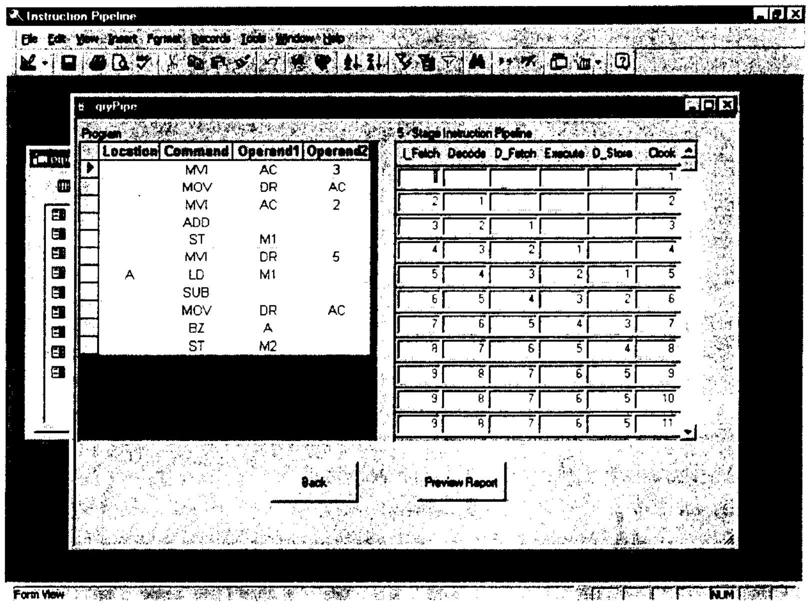
```
       MVI AC,9
       BRA A
       LD M1
       ST M2
    A: MOV DR, AC
       ADD
```

To execute this you separate the pieces of information from the program that match the columns on the entry screen. The program would be divided up like this:

Location	Command	Operand1	Operand2
	MVI	AC	9
	BRA	A	
	LD	M1	
	ST	M2	
A	MOV	DR	AC
	ADD		

Table 2.1

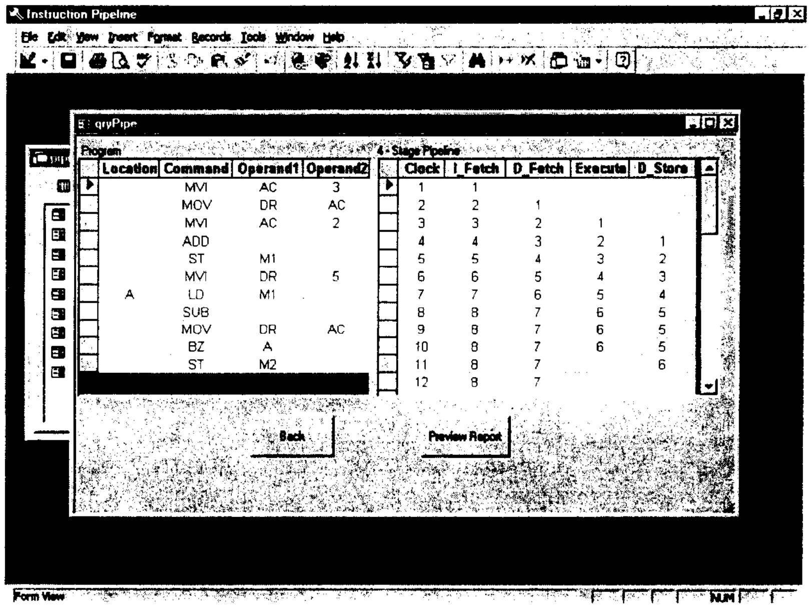
After the program is entered in, you have the option of running a 4 stage instruction pipeline or a 5 stage instruction pipeline. Click the desired option button to the right of the program. When complete hit the "Submit" button to run your pipeline. If you choose a 5 stage pipeline then the "Submit" button will take you to the screen shown in figure 2.4. If you have chosen a 4 stage pipeline then it will take you to the screen in figure 2.5. By hitting the "Back" button you will return to the introductory screen.


Instruction Pipeline
File Edit View Insert Format Records Tools Window Help
qryPipe
Program
5 Stage Instruction Pipeline
Back
Preview Report
Form View
NUM

Location	Command	Operand1	Operand2
	MVI	AC	3
	MOV	DR	AC
	MVI	AC	2
	ADD		
	ST	M1	
	MVI	DR	5
A	LD	M1	
	SUB		
	MOV	DR	AC
	BZ	A	
	ST	M2	

I_Fetch	Decode	D_Fetch	Execute	D_Store	Clock
1					1
2	1				2
3	2	1			3
4	3	2	1		4
5	4	3	2	1	5
6	5	4	3	2	6
7	6	5	4	3	7
8	7	6	5	4	8
9	8	7	6	5	9
9	8	7	6	5	10
9	8	7	6	5	11

Figure 2.4



Program

Location	Command	Operand1	Operand2
	MVI	AC	3
	MOV	DR	AC
	MVI	AC	2
	ADD		
	ST	M1	
	MVI	DR	5
A	LD	M1	
	SUB		
	MOV	DR	AC
	BZ	A	
	ST	M2	

4 - Stage Pipeline

Clock	I_Fetch	D_Fetch	Execute	D_Store
1	1			
2	2	1		
3	3	2	1	
4	4	3	2	1
5	5	4	3	2
6	6	5	4	3
7	7	6	5	4
8	8	7	6	5
9	8	7	6	5
10	8	7	6	5
11	8	7		6
12	8	7		

Figure 2.5

Looking at either figure 2.4 or 2.5, you'll notice that the program that you wrote is on the left of the screen and the pipeline is to the right. The 5 stage pipeline has the following categories: I-Fetch, Decode, D-Fetch, Execute, and D-Store. The 4 stage pipeline combines the I-Fetch and Decode into one.

Commands require different clock times to execute. In this simulation, the commands clock times are 1 for all stages of both pipes for all commands except for the LD and ST commands. The LD command takes 3 clocks in the D-Fetch stage and 1 in the other stages. The ST command takes 3 clocks in the D-Store stage and 1 in the other stages. Each command is assumed to pass through each of the stages regardless if it needs to go to that stage or not. For example, the ADD command must pass through all stages even though it does not store or retrieve data from memory (stages D-Store and D-Fetch, respectively).

The "Back" button returns you to the screen where you enter in the program and select the stage pipeline you want. The "Preview Report" button takes you to the screen that lets you preview the report for printing. The format of the report is different then the screens shown in figure 2.4 and 2.5. It is condensed down to be able to fit on one page (Landscape). To print the instruction pipeline out, hit the "Preview Report" button and then select print (the printer button at the top of the window).

3.0 Testing the Instruction Pipeline Simulation.

To test this simulation, three programs where written to encompass all the possible errors in the logic. An in-depth test on the simulation functionality was not run because the logic is the important part of this and not the functionality. However, I have tried to make the simulation as functionally adequate as possible without running the rigorous tests.

The first test program can be seen in Appendix A. Page 1 of Appendix A shows the 4 and 5 stage pipelines for the program written beneath it. Page 2 and 3 of Appendix A shows the output of the simulation for a 5 stage and a 4 stage pipeline. These two should match up (and they do). This test program was designed to test the MVI, SUB, LD, and ST, commands along with a condition branch jumping forward. Notice that line 4 and line 6 of the program clears the pipeline whether there is a jump or not. This is designed so that a look ahead logic subroutine will not have to be implemented. The I-Fetch of the next commands (commands 5 and 8) will not be able to start until the branch (BZ in this case) has completed executing.

Test program 2 is found in Appendix B. Page 1 is the pipeline and the program. Page 2 and 3 are the simulated pipelines. This program tests out the conditional jump backwards and the ST/LD commands close to each other. When the simulation receives a store command, it decides that a LD command cannot fetch data from memory (D-Fetch) until the final stage of the storing (D-Store) is executing. This is another cautionary step in case the LD command is loading something that is not completely stored. Instruction 7 shows this by holding the LD command in the D-Store pipe for 5 clocks (see the 4 stage pipeline in appendix B). At clock 10 the ST command (instruction 5) is in the last phase of the D-Store stage. This is when the D-Fetch command can start to execute (instruction 7). It takes 3 clocks to complete the D-Fetch stage so it will be in that stage for 5 clocks.

Test program 2 also checks to make sure that the program can branch backwards as well as forward. This is shown in instruct 10 being executed twice. Once jumping back to instruction 7 and the second time continuing on to instruction 11. Notice that in both the computer simulated pipelines and the typed in pipelines that it only goes up to 30 clocks. This is to keep the program simple and to stop incase there is an infinite loop. In the 5 stage pipeline, there wasn't enough room to show the D-Store stage of the ST command (instruction 11) at the end of the program. There is also a limitation to the number of instructions that a person can execute; it is 20. You can type in more than 20 instructions, however, the simulation will only execute the first 20.

Test program 3 is the simplest test program of them all. It's only function is to test the BRA command. In the last two programs a BZ command was used. Since the BRA and BZ commands execute exactly the same (if BZ is TRUE) there didn't seem a need to extensively test it. Only a test of the main functionality of the command needs to be checked. Appendix C holds the pipeline, program and the two computer simulated printouts of test program 3. Again, instruction 2 clears the pipe and the next instruction cannot be fetched until the execution of the command is complete.

4.0 Final Discussion

This simulated pipeline is a simple model of an architecture that probably doesn't have much use in real applications. However, the idea of how a pipeline works and the complexity involved in designing an instruction pipeline was realized from this exercise. I tried to encompass all possible pipeline caveats into the three test programs. However, the program functionality was not completely tested for run time errors. I believe the purpose of this assignment was to evaluate the complexities of an instruction pipeline.

The Appendixes A, B and C hold the test programs and their pipelines. Appendix D holds the main logic code for the pipelines. It may be a bit jumbled. This is due to the lack of knowledge that I have in programming, but I believe it is followable. The reader should be mainly concerned with the modules "readthru4" and "readthru5". This is the main logic for a 4 stage and a 5 stage instruction pipeline.

I deviated a bit from my final proposal of this project. I originally planned on having a variable clock value for each stage. I chose not to do this based on time. When I had first conceived the work involved in this project, I didn't intend on it taking the time that it did. Another change from my proposal was the stages that were available. To me, a 4 and a 5 stage pipeline seemed to make more sense than a 1, 2, or 3 stage pipeline. This is why I decided to do a 4 and 5 stage pipeline.

In the table section of the program, you will notice a "tblProg1", "tblProg2", "tblProg3", and "tblProgram" tables. The simulation looks at the "tblProgram" table as its table to enter in code (or change code). the other three programs refer to the test programs 1, 2 and 3 respectively. If you want to run the test programs, you can either type them in or you can copy the respective table that holds the test program to the "tblProgram" table (this is done by highlighting the test program table, say "tblProg1", and on the tool bar at the top of the window hit Copy, under File, then hit Paste, also under File, and type the name "tblProgram". It will ask you to overwrite it and you hit the "Yes" button.)

Appendix D

```
Private Sub cbSubmit_Click()
  Const X5STAGE = 1
  Const X4STAGE = 2
  Dim stDocName As String
  Dim stDocName2 As String
  Dim stLinkCriteria As String

  DoCmd.SetWarnings False

  If Frame14.Value = X5STAGE Then
    stDocName2 = "qryDelPipe"
    DoCmd.OpenQuery stDocName2, acNormal, acEdit
    stDocName = "frmPipe"
    Call readthru5
 Else
   stDocName2 = "qryDelPipe4"
   DoCmd.OpenQuery stDocName2, acNormal, acEdit
   stDocName = "frmPipe4"
   Call readthru4
 End If
 DoCmd.Close
 DoCmd.OpenForm stDocName, , , stLinkCriteria
 DoCmd.SetWarnings True

End Sub


Sub readthru5()
  Dim db As Database
  Dim tbl As TableDef
' the table that holds the program
  Dim tblProg As TableDef
  Dim rstProg As Recordset
' the pipe table
  Dim tblPipe As TableDef
  Dim rstPipe As Recordset
' array of command line information
  Dim locArray(1 To 30) As String
  Dim comArray(1 To 30) As String
  Dim oper1Array(1 To 30) As Variant
  Dim oper2Array(1 To 30) As Variant
' variables need to keep track of
  Dim AC As Integer  ' accumulator register
  Dim DR As Integer  ' data register
  Dim M1 As Integer  ' memery location 1
  Dim M2 As Integer  ' memory location 2

' the stages of the pipe
  Dim iFetch(1 To 40) As Integer
  Dim Decode(1 To 40) As Integer
  Dim dFetch(1 To 40) As Integer
  Dim Execute(1 To 40) As Integer
  Dim dStore(1 To 40) As Integer
'the position that each stage is on
  Dim posIF As Integer
  Dim posDC As Integer
  Dim posDF As Integer
  Dim posEX As Integer
  Dim posDS As Integer
  Dim pgmCnt As Integer
' a holder for the current command
  Dim currentCmd As String
' used to exit looking at program loop
  Dim FinProg As Boolean
' used to tell when branching is allowed
  Dim Branch As Boolean
```

```
' dummy variables
  Dim finLoop As Boolean
  Dim i As Integer
  Dim message As String
  Dim DEBUG_PO As Integer

' initialize all need variables
DEBUG_PO = 0  'for debugging using print out statements
Set db = CurrentDb
Set tblProg = db.TableDefs("tblProgram")
Set rstProg = tblProg.OpenRecordset(dbOpenTable)
' Set rstProg = tblProg.OpenRecordset(dbOpenDynaset)
AC = 0
DR = 0
M1 = 0
M2 = 0
posIF = 1
posDC = 2
posDF = 3
posEX = 4
posDS = 5
On Error Resume Next
For i = 1 To 30
  iFetch(i) = 0
  Decode(i) = 0
  dFetch(i) = 0
  Execute(i) = 0
  dStore(i) = 0
  locArray(i) = Null
  comArray(i) = Null
  oper1Array(i) = Null
  oper2Array(i) = Null
Next
On Error GoTo 0
FinProg = False

' check file to make sure it is not empty
rstProg.MoveFirst
If rstProg.EOF Then
 MsgBox "EOF error on tblProgram"
 Exit Sub
End If
pgmCnt = 1
' put commands in command array
Do While Not rstProg.EOF
  On Error Resume Next
    locArray(pgmCnt) = rstProg.Fields("Location")
    comArray(pgmCnt) = rstProg.Fields("Command")
    oper1Array(pgmCnt) = rstProg.Fields("Operand1")
    oper2Array(pgmCnt) = rstProg.Fields("Operand2")
  On Error GoTo 0
  pgmCnt = pgmCnt + 1
  rstProg.MoveNext
'  message = "The assignment loop - comArray: " & comArray(pgmCnt) & Chr(13) _
        & "Location: " & locArray(pgmCnt) & Chr(13) & "Operand1: " & _
        oper1Array(pgmCnt)
'  MsgBox (message)
Loop

' loop through commands until end of program or 30 clks have ellapsed
pgmCnt = 1
Do While Not FinProg
'  message = "The programming loop - comArray: " & comArray(pgmCnt)
'  MsgBox (message)
 currentCmd = comArray(pgmCnt)
 If DEBUG_PO = 1 Then
   Call PrintOut2("CurrentCmd:", currentCmd)
 End If
 Select Case currentCmd
   Case "ADD"
   iFetch(posIF) = pgmCnt
   Decode(posDC) = pgmCnt
   dFetch(posDF) = pgmCnt
   Execute(posEX) = pgmCnt
   dStore(posDS) = pgmCnt
   posIF = posDC
   posDC = posDF
   posDF = posEX
   posEX = posDS
   posDS = posDS + 1
   pgmCnt = pgmCnt + 1
   If DEBUG_PO = 1 Then
     Call PrintOut4("ADD - AC:", AC, "DR:", DR)
   End If
   AC = AC + DR
 Case "SUB"
   iFetch(posIF) = pgmCnt
   Decode(posDC) = pgmCnt
   dFetch(posDF) = pgmCnt
   Execute(posEX) = pgmCnt
   dStore(posDS) = pgmCnt
   posIF = posDC
   posDC = posDF
   posDF = posEX
   posEX = posDS
   posDS = posDS + 1
   pgmCnt = pgmCnt + 1
   If DEBUG_PO = 1 Then
     Call PrintOut4("SUB - AC:", AC, "DR:", DR)
   End If
   AC = AC - DR
 Case "MVI"
   iFetch(posIF) = pgmCnt
   Decode(posDC) = pgmCnt
   dFetch(posDF) = pgmCnt
   Execute(posEX) = pgmCnt
   dStore(posDS) = pgmCnt
   posIF = posDC
   posDC = posDF
   posDF = posEX
   posEX = posDS
   posDS = posDS + 1
   If DEBUG_PO = 1 Then
     Call PrintOut4("MVI - Oper1:", oper1Array(pgmCnt), "Oper2:", oper2Array(pgmCnt))
   End If
   If oper1Array(pgmCnt) = "AC" Then
     AC = oper2Array(pgmCnt)
   ElseIf oper1Array(pgmCnt) = "DR" Then
     DR = oper2Array(pgmCnt)
   End If
   If DEBUG_PO = 1 Then
     Call PrintOut4("AC:", AC, "DR:", DR)
   End If
   pgmCnt = pgmCnt + 1
 Case "MOV"
   iFetch(posIF) = pgmCnt
   Decode(posDC) = pgmCnt
   dFetch(posDF) = pgmCnt
   Execute(posEX) = pgmCnt
   dStore(posDS) = pgmCnt
   posIF = posDC
   posDC = posDF
   posDF = posEX
   posEX = posDS
   posDS = posDS + 1
   If oper1Array(pgmCnt) = "DR" Then
     DR = AC
   ElseIf oper1Array(pgmCnt) = "AC" Then
     AC = DR
   End If
```

```
    If DEBUG_PO = 1 Then
      Call PrintOut4("AC:", AC, "DR:", DR)
    End If
    pgmCnt = pgmCnt + 1
  Case "LD"
    iFetch(posIF) = pgmCnt
    Decode(posDC) = pgmCnt
    dFetch(posDF) = pgmCnt
    dFetch(posDF + 1) = pgmCnt
    dFetch(posDF + 2) = pgmCnt
    Execute(posEX + 2) = pgmCnt
    dStore(posDS + 2) = pgmCnt
    ' remove 0's for blank stage
    For i = 0 To 1
      Execute(posEX + i) = -1
      dStore(posDS + i) = -1
    Next
    posIF = posDC
    posDC = posDF + 2
    posDF = posDF + 3
    posEX = posEX + 3
    posDS = posDS + 3
    If oper1Array(pgmCnt) = "M1" Then
      AC = M1
    ElseIf oper1Array(pgmCnt) = "M2" Then
      AC = M2
    End If
    pgmCnt = pgmCnt + 1
    If DEBUG_PO = 1 Then
      Call PrintOut4("M1: ", M1, "M2: ", M2)
    End If
  Case "ST"
    iFetch(posIF) = pgmCnt
    Decode(posDC) = pgmCnt
    dFetch(posDF) = pgmCnt
    Execute(posEX) = pgmCnt
    dStore(posDS) = pgmCnt
    dStore(posDS + 1) = pgmCnt
    dStore(posDS + 2) = pgmCnt
    posIF = posDC
    posDC = posDF
    posDF = posEX
    posEX = posDS + 2
    posDS = posDS + 3
    If oper1Array(pgmCnt) = "M1" Then
      M1 = AC
    ElseIf oper1Array(pgmCnt) = "M2" Then
      M2 = AC
    End If
    If DEBUG_PO = 1 Then
      Call PrintOut2("Operand1: ", oper1Array(pgmCnt))
      Call PrintOut4("M1: ", M1, "M2: ", M2)
    End If
    pgmCnt = pgmCnt + 1
  Case "BRA", "BZ"
    iFetch(posIF) = pgmCnt
    Decode(posDC) = pgmCnt
    dFetch(posDF) = pgmCnt
    Execute(posEX) = pgmCnt
    dStore(posDS) = pgmCnt
    ' put in null char to simulate clearing the pipe
    For i = posIF + 1 To posDS - 1
      iFetch(i) = -1
    Next
    For i = posDC + 1 To posDS
      Decode(i) = -1
    Next
    For i = posDF + 1 To posDS + 1
      dFetch(i) = -1
    Next
```

```
        For i = posEX + 1 To posDS + 2
          Execute(i) = -1
        Next
        For i = posDS + 1 To posDS + 3
          dStore(i) = -1
        Next
        ' reasign new positions
        posIF = posDS
        posDC = posDS + 1
        posDF = posDS + 2
        posEX = posDS + 3
        posDS = posDS + 4
        If AC = 0 Then
          Branch = True
        ElseIf comArray(pgmCnt) = "BRA" Then
          Branch = True
        Else
          Branch = False
        End If
        If DEBUG_PO = 1 Then
          Call PrintOut4("AC:", AC, "comArray(pgmCnt):", comArray(pgmCnt))
        End If
        If Branch = True Then
          If DEBUG_PO = 1 Then
            message = "oper1Array(pgmCnt) = " & oper1Array(pgmCnt)
            MsgBox (message)
          End If
          If oper1Array(pgmCnt) = Null Then
            pgmCnt = pgmCnt + 1
          Else
          ' search for operand to match location and set pgmCnt to that command count
            i = 0
            finLoop = False
            Do While Not finLoop
              i = i + 1
              If DEBUG_PO = 1 Then
                message = "i: " & i & "   pgmCnt = " & pgmCnt & Chr(13) _
                   & "locArray(i) = " & locArray(i) & Chr(13) _
                   & "oper1Array(pgmCnt) = " & oper1Array(pgmCnt)
                MsgBox (message)
              End If
              If locArray(i) = oper1Array(pgmCnt) Then
                pgmCnt = i
                finLoop = True
              ElseIf i = 30 Then
                finLoop = True
              End If
            Loop
            If Not (pgmCnt = i) Then
              pgmCnt = pgmCnt + 1
            End If
          End If
        Else
          pgmCnt = pgmCnt + 1
        End If
      Case Else
        FinProg = True
    End Select

    If posDS > 32 Or pgmCnt > 20 Then
      FinProg = True
    End If
  Loop

  ' fill in the blanks
  For i = posIF - 1 To 2 Step -1
    If iFetch(i) = 0 Then
      iFetch(i) = iFetch(i + 1)
    End If
  Next
```

```
For i = posDC - 1 To 3 Step -1
  If Decode(i) = 0 Then
    Decode(i) = Decode(i + 1)
  End If
Next
For i = posDF - 1 To 4 Step -1
  If dFetch(i) = 0 Then
    dFetch(i) = dFetch(i + 1)
  End If
Next
For i = posEX - 1 To 5 Step -1
  If Execute(i) = 0 Then
    Execute(i) = Execute(i + 1)
  End If
Next
For i = posDS - 1 To 6 Step -1
  If dStore(i) = 0 Then
    dStore(i) = dStore(i + 1)
  End If
Next

' put into table
Set tblPipe = db.TableDefs("tblPipe")
Set rstPipe = tblPipe.OpenRecordset(dbOpenDynaset)

For i = 1 To 30
  rstPipe.AddNew
  rstPipe.Fields("Clock") = i  ' set the clock time
  If iFetch(i) > 0 Then
    rstPipe.Fields("I_Fetch") = iFetch(i)
  Else
    rstPipe.Fields("I_Fetch") = Null
  End If
  If Decode(i) > 0 Then
    rstPipe.Fields("Decode") = Decode(i)
  Else
    rstPipe.Fields("Decode") = Null
  End If
  If dFetch(i) > 0 Then
    rstPipe.Fields("D_Fetch") = dFetch(i)
  Else
    rstPipe.Fields("D_Fetch") = Null
  End If
  If Execute(i) > 0 Then
    rstPipe.Fields("Execute") = Execute(i)
  Else
    rstPipe.Fields("Execute") = Null
  End If
  If dStore(i) > 0 Then
    rstPipe.Fields("D_Store") = dStore(i)
  Else
    rstPipe.Fields("D_Store") = Null
  End If
  rstPipe.Update
Next
MsgBox ("Pipeline Completed.")
End Sub
```

Sub PrintOut2(x As Variant, y As Variant)
```
  Dim message As String
  message = x & "   " & y
  MsgBox (message)
End Sub
```
Sub PrintOut4(x As Variant, y As Variant, w As Variant, z As Variant)
```
  Dim message As String
  message = x & "   " & y & Chr(13) & w & "   " & z
  MsgBox (message)
End Sub
```

```
Sub readthru4()
  Dim db As Database
  Dim tbl As TableDef
' the table that holds the program
  Dim tblProg As TableDef
  Dim rstProg As Recordset
' the pipe table
  Dim tblPipe As TableDef
  Dim rstPipe As Recordset
' array of command line information
  Dim locArray(1 To 30) As String
  Dim comArray(1 To 30) As String
  Dim oper1Array(1 To 30) As Variant
  Dim oper2Array(1 To 30) As Variant
' variables need to keep track of
  Dim AC As Integer  ' accumulator register
  Dim DR As Integer  ' data register
  Dim M1 As Integer  ' memery location 1
  Dim M2 As Integer  ' memory location 2

' the stages of the pipe
  Dim iFetch(1 To 40) As Integer
  Dim dFetch(1 To 40) As Integer
  Dim Execute(1 To 40) As Integer
  Dim dStore(1 To 40) As Integer
'the position that each stage is on
  Dim posIF As Integer
  Dim posDF As Integer
  Dim posEX As Integer
  Dim posDS As Integer
  Dim pgmCnt As Integer
' a holder for the current command
  Dim currentCmd As String
' used to exit looking at program loop
  Dim FinProg As Boolean
' used to tell when branching is allowed
  Dim Branch As Boolean
' dummy variables
  Dim finLoop As Boolean
  Dim i As Integer
  Dim message As String

' initialize all need variables
Set db = CurrentDb
Set tblProg = db.TableDefs("tblProgram")
Set rstProg = tblProg.OpenRecordset(dbOpenTable)
' Set rstProg = tblProg.OpenRecordset(dbOpenDynaset)
AC = 0
DR = 0
M1 = 0
M2 = 0
posIF = 1
posDF = 2
posEX = 3
posDS = 4
On Error Resume Next
For i = 1 To 30
  iFetch(i) = 0
  dFetch(i) = 0
  Execute(i) = 0
  dStore(i) = 0
  locArray(i) = Null
  comArray(i) = Null
  oper1Array(i) = Null
  oper2Array(i) = Null
Next
On Error GoTo 0
FinProg = False

' check file to make sure it is not empty
```

```
rstProg.MoveFirst
If rstProg.EOF Then
 MsgBox "EOF error on tblProgram"
 Exit Sub
End If
pgmCnt = 1
' put commands in command array
Do While Not rstProg.EOF
  On Error Resume Next
    locArray(pgmCnt) = rstProg.Fields("Location")
    comArray(pgmCnt) = rstProg.Fields("Command")
    oper1Array(pgmCnt) = rstProg.Fields("Operand1")
    oper2Array(pgmCnt) = rstProg.Fields("Operand2")
  On Error GoTo 0
  pgmCnt = pgmCnt + 1
  rstProg.MoveNext
'  message = "The assignment loop - comArray: " & comArray(pgmCnt) & Chr(13) _
        & "Location: " & locArray(pgmCnt) & Chr(13) & "Operand1: " & _
        oper1Array(pgmCnt)
'  MsgBox (message)
Loop

' loop through commands until end of program or 30 clks have ellapsed
pgmCnt = 1
Do While Not FinProg
'  message = "The programming loop - comArray: " & comArray(pgmCnt)
'  MsgBox (message)
  currentCmd = comArray(pgmCnt)
'  Call PrintOut2("CurrentCmd:", currentCmd)
  Select Case currentCmd
    Case "ADD"
      iFetch(posIF) = pgmCnt
      dFetch(posDF) = pgmCnt
      Execute(posEX) = pgmCnt
      dStore(posDS) = pgmCnt
      posIF = posDF
      posDF = posEX
      posEX = posDS
      posDS = posDS + 1
      pgmCnt = pgmCnt + 1
     'Call PrintOut4("AC:", AC, "DR:", DR)
      AC = AC + DR
    Case "SUB"
      iFetch(posIF) = pgmCnt
      dFetch(posDF) = pgmCnt
      Execute(posEX) = pgmCnt
      dStore(posDS) = pgmCnt
      posIF = posDF
      posDF = posEX
      posEX = posDS
      posDS = posDS + 1
      pgmCnt = pgmCnt + 1
     'Call PrintOut4("AC:", AC, "DR:", DR)
      AC = AC - DR
    Case "MVI"
      iFetch(posIF) = pgmCnt
      dFetch(posDF) = pgmCnt
      Execute(posEX) = pgmCnt
      dStore(posDS) = pgmCnt
      posIF = posDF
      posDF = posEX
      posEX = posDS
      posDS = posDS + 1
     'Call PrintOut4("Oper1:", oper1Array(pgmCnt), "Oper2:", oper2Array(pgmCnt))
      If oper1Array(pgmCnt) = "AC" Then
        AC = oper2Array(pgmCnt)
      ElseIf oper1Array(pgmCnt) = "DR" Then
        DR = oper2Array(pgmCnt)
      End If
     'Call PrintOut4("AC:", AC, "DR:", DR)
    pgmCnt = pgmCnt + 1
  Case "MOV"
    iFetch(posIF) = pgmCnt
    dFetch(posDF) = pgmCnt
    Execute(posEX) = pgmCnt
    dStore(posDS) = pgmCnt
    posIF = posDF
    posDF = posEX
    posEX = posDS
    posDS = posDS + 1
    If oper1Array(pgmCnt) = "DR" Then
      DR = AC
    ElseIf oper1Array(pgmCnt) = "AC" Then
      AC = DR
    End If
   ' Call PrintOut4("AC:", AC, "DR:", DR)
    pgmCnt = pgmCnt + 1
  Case "LD"
    iFetch(posIF) = pgmCnt
    dFetch(posDF) = pgmCnt
    dFetch(posDF + 1) = pgmCnt
    dFetch(posDF + 2) = pgmCnt
    Execute(posEX + 2) = pgmCnt
    dStore(posDS + 2) = pgmCnt
    ' remove 0's for blank stage
    For i = 0 To 1
      Execute(posEX + i) = -1
      dStore(posDS + i) = -1
    Next
    posIF = posDF + 2
    posDF = posDF + 3
    posEX = posEX + 3
    posDS = posDS + 3
    If oper1Array(pgmCnt) = "M1" Then
      AC = M1
    ElseIf oper1Array(pgmCnt) = "M2" Then
      AC = M2
    End If
    pgmCnt = pgmCnt + 1
  Case "ST"
    iFetch(posIF) = pgmCnt
    dFetch(posDF) = pgmCnt
    Execute(posEX) = pgmCnt
    dStore(posDS) = pgmCnt
    dStore(posDS + 1) = pgmCnt
    dStore(posDS + 2) = pgmCnt
    posIF = posDF
    posDF = posEX
    posEX = posDS + 2
    posDS = posDS + 3
    If oper1Array(pgmCnt) = "M1" Then
      M1 = AC
    ElseIf oper1Array(pgmCnt) = "M2" Then
      M2 = AC
    End If
    pgmCnt = pgmCnt + 1
  Case "BRA", "BZ"
    iFetch(posIF) = pgmCnt
    dFetch(posDF) = pgmCnt
    Execute(posEX) = pgmCnt
    dStore(posDS) = pgmCnt
    ' put in null char to simulate clearing the pipe
    For i = posIF + 1 To posDS - 1
      iFetch(i) = -1
    Next
    For i = posDF + 1 To posDS + 1
      dFetch(i) = -1
    Next
    For i = posEX + 1 To posDS + 2
      Execute(i) = -1
```

```
        Next
        For i = posDS + 1 To posDS + 3
          dStore(i) = -1
        Next
        ' reasign new positions
        posIF = posDS
        posDF = posDS + 1
        posEX = posDS + 2
        posDS = posDS + 3
        If AC = 0 Then
          Branch = True
        ElseIf comArray(pgmCnt) = "BRA" Then
          Branch = True
        Else
          Branch = False
        End If
       ' Call PrintOut4("AC:", AC, "comArray(pgmCnt):", comArray(pgmCnt))
        If Branch = True Then
       '   message = "oper1Array(pgmCnt) = " & oper1Array(pgmCnt)
       '   MsgBox (message)
          If oper1Array(pgmCnt) = Null Then
            pgmCnt = pgmCnt + 1
          Else
          ' search for operand to match location and set pgmCnt to that command count
            i = 0
            finLoop = False
            Do While Not finLoop
              i = i + 1
           '   message = "i: " & i & "   pgmCnt = " & pgmCnt & Chr(13) _
                  & "locArray(i) = " & locArray(i) & Chr(13) _
                  & "oper1Array(pgmCnt) = " & oper1Array(pgmCnt)
           '   MsgBox (message)
              If locArray(i) = oper1Array(pgmCnt) Then
                pgmCnt = i
                finLoop = True
              ElseIf i = 30 Then
                finLoop = True
              End If
            Loop
            If Not (pgmCnt = i) Then
              pgmCnt = pgmCnt + 1
            End If
          End If
        Else
          pgmCnt = pgmCnt + 1
        End If
      Case Else
        FinProg = True
    End Select

    If posDS > 32 Or pgmCnt > 20 Then
      FinProg = True
    End If
  Loop

  ' fill in the blanks
  For i = posIF - 1 To 2 Step -1
    If iFetch(i) = 0 Then
      iFetch(i) = iFetch(i + 1)
    End If
  Next
  For i = posDF - 1 To 4 Step -1
    If dFetch(i) = 0 Then
      dFetch(i) = dFetch(i + 1)
    End If
  Next
  For i = posEX - 1 To 5 Step -1
    If Execute(i) = 0 Then
      Execute(i) = Execute(i + 1)
    End If
```

```
Next
For i = posDS - 1 To 6 Step -1
  If dStore(i) = 0 Then
    dStore(i) = dStore(i + 1)
  End If
Next

' put into table
Set tblPipe = db.TableDefs("tblPipe4")
Set rstPipe = tblPipe.OpenRecordset(dbOpenDynaset)

For i = 1 To 30
  rstPipe.AddNew
  rstPipe.Fields("Clock") = i  ' set the clock time
  If iFetch(i) > 0 Then
    rstPipe.Fields("I_Fetch") = iFetch(i)
  Else
    rstPipe.Fields("I_Fetch") = Null
  End If
  If dFetch(i) > 0 Then
    rstPipe.Fields("D_Fetch") = dFetch(i)
  Else
    rstPipe.Fields("D_Fetch") = Null
  End If
  If Execute(i) > 0 Then
    rstPipe.Fields("Execute") = Execute(i)
  Else
    rstPipe.Fields("Execute") = Null
  End If
  If dStore(i) > 0 Then
    rstPipe.Fields("D_Store") = dStore(i)
  Else
    rstPipe.Fields("D_Store") = Null
  End If
  rstPipe.Update
Next
MsgBox ("Pipeline Completed.")

End Sub
```
Private Sub Command24_Click()
```
On Error GoTo Err_Command24_Click

   Dim stDocName As String
   Dim stLinkCriteria As String

   stDocName = "frmPipe4"
   DoCmd.OpenForm stDocName, , , stLinkCriteria

Exit_Command24_Click:
   Exit Sub

Err_Command24_Click:
   MsgBox Err.Description
   Resume Exit_Command24_Click

End Sub
```
Private Sub Command25_Click()
```
On Error GoTo Err_Command25_Click

   Dim stDocName As String

   stDocName = "qryDelPipe"
   DoCmd.OpenQuery stDocName, acNormal, acEdit

Exit_Command25_Click:
   Exit Sub

Err_Command25_Click:
   MsgBox Err.Description
   Resume Exit_Command25_Click
```

```
End Sub

Private Sub Command27_Click()
On Error GoTo Err_Command27_Click

    Dim stDocName As String
    Dim stLinkCriteria As String

    DoCmd.Close

    stDocName = "frmIntro1"
    DoCmd.OpenForm stDocName, , , stLinkCriteria

Exit_Command27_Click:
    Exit Sub

Err_Command27_Click:
    MsgBox Err.Description
    Resume Exit_Command27_Click

End Sub
```